AMENDMENTS TO THE SPECIFICATION

Please amend the following paragraph 007 page 2:

[0007] These and other features, aspects, and advantages of the present invention are better understood when the following Detailed Description of the Invention is read with reference to the accompanying drawings, wherein:

FIG. 1 depicts a possible operating environment for one embodiment of the present invention;

FIG. 2 is a block diagram of a microprocessor;

FIG~~S~~. 3 ~~and 4 are~~ is a block diagram~~s~~ of a microprocessor pipeline;

FIG. ~~5~~ 4 is a block diagram of an instruction bundle;

FIG. ~~6~~ 5 A and B are is a block diagrams illustrating one embodiment of the present invention;

FIG. ~~7~~ 6 is a block diagram illustrating the execution of a complex microprocessor instruction; and

FIGS. ~~8~~ 7A and 7B are ~~is a~~ block diagrams illustrating another embodiment of the present invention.

Please amend paragraph 0015 page 5 as follows:

[0015] FIG~~S~~. 3 ~~and 4 are~~ is a block diagram~~s~~ of a nine-stage pipeline. FIG. 3 is a simplified block diagram showing an integer pipeline 58 and a floating-point pipeline 60. ~~FIG. 4 is a detailed block diagram of the pipeline stages.~~ An instruction to the microprocessor (shown as reference numeral 10 in FIGS. 1 and 2) advances through the integer pipeline 58 and the floating-point pipeline 60 in one of these stages. The integer pipeline 58 has three additional stages, N_1, N_2, and N_3. These additional stages make the integer pipeline 58 symmetrical with the

floating point pipeline 60. Because the general concept of a pipelined microprocessor has been known for over ten (10) years, the stages are only briefly described. The nine stages of the integer pipeline 58 include a fetch stage 62, a decode stage 64, a grouping stage 66, an execution stage 68, a cache access stage 70, a miss/hit stage 72, an executed floating point instruction stage 74, a trap stage 76, and a write stage 78. The floating-point pipeline 60 has a register stage 80 and execution stages X_1, X_2, and X_3 (shown as reference numeral 82). Prior to an instruction being executed, the instruction is fetched from the instruction cache unit (shown as reference numeral 36 in FIG. 3) and placed in the instruction buffer (shown as reference numeral 44 in FIG. 2). Because the Prefetch and Dispatch Unit (shown as reference numeral 42 in FIG. 2) may also predict an instruction to speed processing, a predicted instruction is also stored in the instruction buffer. The decode stage 64 retrieves a fetched instruction stored in the instruction buffer, pre-decodes the fetched instruction, and then return stores pre-decoded bits in the instruction buffer. The grouping stage *66* receives, groups, and dispatches one or more valid instructions per cycle. The grouping stage *66,* for example, could receive four (4) valid instructions from the Prefetch and Dispatch Unit. Up to two (2) floating-point instructions, or two (2) graphics instructions, from the four valid candidates could be sent to the Floating Point Unit and/or to the Graphics Unit (shown respectively as reference numerals 52 and 54 in FIG. 2).

Please amend the following paragraphs starting at 0019 page 7:

[0019] FIG. ~~5~~ 4 is a block diagram of an instruction bundle 84. The instruction bundle 84 comprises eight (8) instructions that are fetched from the instruction cache unit 36. Superscalar microprocessor designs, such as the microprocessor 10 shown in FIG. 2, achieve high performance by executing multiple instructions per clock cycle. Because multiple instructions are executed per clock cycle, the instruction cache unit 36 fetches multiple instructions during each clock cycle. The term "clock cycle," as used herein, refers to an interval of time accorded to various stages of the instruction processing pipeline within the microprocessor.

[0020] As FIG. ~~5~~ 4 shows, the instruction bundle 84 may comprise valid instructions 86, complex instructions 88, and invalid instructions 90. Each instruction has an associated valid bit and an error bit. When the valid bit is set high (or "1"), the instruction associated with that valid bit is recognized as a valid instruction. When the error bit, conversely, is set high (and thus the valid bit is set low or "0"), the instruction associated with that error bit is invalid. The complex instruction 88 is a more complex instruction that is executed by hardware. The complex instruction 88 contains helper instructions — these helper instructions require more hardware tasks, so the helper instructions are broken down into smaller instructions and then executed. Even though the instruction bundle 84 may contain valid instructions 86, complex instructions 88, and invalid instructions 90, these instructions are guaranteed to be monotonically valid. "Monotonically" valid means that all the valid instructions 86, having their respective valid bit set high (or "1"), are at the front, or "top," of the instruction bundle 84. any invalid instructions 90 having their respective valid bit set low (or "0"), are at the back, or the "bottom," of the instruction bundle 84. The number of valid instructions within the bundle is necessary, for a computer system's resources are allocated based upon the number of valid instructions.

[0021] FIGS. ~~6~~ 5A and B are ~~is a~~ block diagrams illustrating one embodiment of the present invention for determining the number of valid instructions 86 within the instruction bundle 84. When the complex instruction 88 is detected, the original instruction bundle 84 is broken at the instruction prior to the complex instruction 88. FIG. ~~6~~ 5A shows, therefore, the original instruction bundle 84 is broken at instruction #2. Instructions #1 and #2 are treated as a first new instruction bundle 92, shown in FIG. ~~6~~ 5B, with all other instructions in the first new instruction bundle 92 marked as invalid instructions 90. Because the first new instruction bundle 92 is monotonic — that is, all the valid instructions, and corresponding valid bits, are compressed at the "top" of the bundle 92 — the number of valid instructions in the bundle 92 may be edge detected. An edge detection circuit may be used to detect when the string of valid bits, corresponding to each valid instructions, transitions from high ("1") to low ("0"). The monotonic nature of the bundle ensures any valid instructions will lie from the first instructions

slot #1 and onward. Once an invalid instruction is encountered, every instruction afterwards will be invalid. The edge detect circuit, therefore, detects a "1" to "0" transition and stops — there's no need to population count the number of valid bits in each slot in the bundle. During a first clock cycle, therefore, the valid instructions #1 and #2, of the first new instruction bundle 92, are sent for execution along the pipeline.

[0022] FIG. ~~7~~ 6 is a block diagram illustrating the execution of the complex instruction 88. With the valid instructions #1 and #2, of the first new instruction bundle 92, sent for execution during the first clock cycle, the complex instruction 88, with its helper instructions, is sent for execution during a next second clock cycle. Once these helpers instructions are executed, the remaining instructions #4-#8, in the original instruction bundle 84, must be sent for execution.

[0023] FIGS. ~~8~~ 7 ~~is a~~ are block diagrams illustrating another embodiment of the present invention for determining the number of valid instructions within an instruction bundle. FIG. ~~8~~ 7A shows what remains of the original instruction bundle (shown as reference numeral 84 in FIGS. ~~5-7~~ 4-6), while FIG. ~~8-~~7B shows a shifted instruction bundle. The original instruction bundle, to recap, was broken to form the first new instruction bundle (shown as reference numeral 92 in FIG. ~~6-~~5B). The valid instructions of the first new instruction bundle, previously occupying instruction slots #1 and #2, were sent for execution during the first clock cycle. The complex instruction (shown as reference numeral 88 in FIGS. ~~5-7~~ 4-6), previously occupying instruction slot #3, was sent for execution during the second clock cycle. Thus the first three instruction slots #1-#3, within the original instruction bundle, have been sent for execution during the first two clock cycles. FIG. ~~8-~~7A shows that what remains of the original instruction bundle, now termed the remaining instruction bundle 94, is no longer monotonic — that is, the valid instructions 86 are sparsely populated within the remaining instruction bundle 94. Because the number of valid instructions within the remaining instruction bundle 94 must again be determined to allocate system resources, an edge detection circuit may again be used.

[0024] FIGS. ~~8~~ 7A and ~~8~~ 7B show the valid instructions 86 may be shifted to form a monotonic bundle. Because the instructions in slots #1-#3 were sent for execution during the first two clock cycles, the remaining valid instructions 86 may be shifted up to the top of the bundle. This shifting process produces a shifted instruction bundle 96 shown in FIG. ~~8~~ 7B. Notice however, that this shifting process again ensures a monotonic arrangement — all the valid instructions 86, and their corresponding valid bits, are shifted to the "top" of the shifted instruction bundle 96. The number of valid instructions in the shifted instruction bundle 96 may then be determined with an edge detection circuit. Edge detecting the valid bit transitions from high ("1") to low ("0") allows the number of valid instructions to be quickly determined. Because the shifted instruction bundle 96 is now monotonic, once an invalid instruction is encountered, every instruction afterwards will be invalid. There is no need to population count the number of valid bits within each slot in the shifted instruction bundle 96.

[0025] Timing slack permits the shifting and edge detection of the valid instructions. Because the instructions in slots #1-#3 were sent for execution during the first two clock cycles, the remaining valid instructions the original instruction bundle (shown as reference numeral 84 in FIG. ~~5-7~~ 4-6) are not sent for execution until the third clock cycle. The shifted instruction bundle 96, in other words, is not sent for execution until after the valid instructions 86, prior to the complex instruction 88, are sent for execution during the first clock cycle, and until after the complex instruction 88 is sent for execution during the second clock cycle. Thus the bundling of the valid instructions occurring prior to the complex instruction 88, and the bundling of the helper instructions, creates timing slack that allows shifting and edge detecting the remaining valid instructions in the shifted instruction bundle 96.